

Chapter 7

Testing and Evaluation

Learning Objectives

At the conclusion of this chapter, the reader will be able to:

- Administer a formal testing methodology (e.g., unit test, integrated test, stress test, acceptance test)
- Implement and monitor compliance with internal controls to protect resources and ensure availability, confidentiality and integrity during testing (e.g., security audits, versioning control, change control)
- Validate implementations against contractual terms and design specifications
- Evaluate that expected benefits are achieved and report metrics (e.g., return on investment, benchmarks, user satisfaction)

Introduction

Healthcare organizations rely on information systems to manage clinical, administrative, financial and legal aspects of daily operations. As technology advances and new healthcare systems are developed and marketed, stakeholders must weigh the risks of implementing or modifying systems and mitigate those risks as much as possible. A critical element of that risk-mitigation strategy is testing and evaluating any new or modified component or system.

Purpose of Systems Testing

The fundamental purpose of system testing is to provide knowledge to assist in managing the risks involved in developing, producing, operating and sustaining systems and their capabilities. Specifically, system testing provides knowledge

of capabilities and limitations to the stakeholders for use in improving the system performance, and to the user community for optimizing system use and sustaining operations. Furthermore, system testing identifies the technical and operational limitations of the system under development so they can be resolved prior to production and deployment.¹ Information systems have become an integral part of healthcare operations and, as such, often have a direct impact on patient safety. With this in mind, identifying and testing for areas that are likely to be major patient safety risks is very important.

System testing and evaluation are performed on both hardware and software. Hardware testing includes evaluation of the physical components of the system (e.g., circuits, drives, internal components, etc.). Software testing is an investigation of the quality and validation of the functionality of a software product or service with the goal of finding any defects or “bugs” and fixing them before the product is released. Software testing can also provide an objective, independent view of the software that enables the business to appreciate and understand the risks of implementation. Test techniques include, but are not limited to, executing a program or application with the intent of finding software errors or other defects. Comprehensive testing is a process of validating and verifying that a system:

- Meets the requirements that guided its design and development
- Responds correctly to all kinds of inputs
- Performs its functions within an acceptable time
- Is sufficiently usable
- Can be implemented and run in its intended environments
- Achieves the general results the stakeholders desire²

Appropriate testing is critical for the success of any new or upgraded system. A study conducted by the National Institute of Standards and Technology (NIST) released in 2002 reported that software bugs (coding issues as well as integration challenges) cost the U.S. economy \$59.5 billion annually.³ It also found that more than a third of these costs could have been avoided if better testing was performed to enable earlier and more effective identification and resolution of defects; the earlier a defect is found within the product development life cycle, the cheaper it is to fix. In 2017, the estimated cumulative cost of software bugs and failures worldwide grew to \$1.7 trillion, affecting at least 3.7 billion people.⁴

The first step in executing a successful test is creating a test methodology.

Test Methodology

Different types of methodologies are used in the field of systems testing and quality assurance for today’s complex healthcare information technology (IT) systems. Whether testing an enterprise-level system, an individual workflow or

application, or a specific piece of code, the methodology is equally important. Due to the complexity of healthcare systems, many healthcare organizations adopt a buy-not-build strategy, outsourcing development to or purchasing off-the-shelf solutions from companies that specialize in that area. As a result, their IT staff often has only a partial picture of their system's development history. In such cases, a well-defined testing methodology is critical to ensure that the delivered system meets the needs of the healthcare organization. Testing methodologies are the strategies and approaches used to test a particular product to ensure it is fit for purpose. Testing methodologies usually involve testing that the product works in accordance with its specification, has no undesirable side effects when used in ways outside of its design parameters, and, in a worst case, will fail safely.⁵ Testing scenarios vary widely among healthcare systems and are tailored for each organization by the test teams and stakeholders, but a sound testing methodology generally includes the following key steps:

- Define the test strategy
- Develop testing tools
- Execute testing
- Employ test controls
- Report on testing results
- Perform final evaluation

Each of these steps will be discussed in further detail in the following sections.

Test Strategy

The test strategy is a formal, high-level description of how a system will be tested. It is developed in order to address all facets of the testing process and ensure testing objectives are achieved. The test strategy, also referred to as the test approach, may include testing scope and objectives, current business issues to consider, testing roles and responsibilities, status reporting methods, test automation and tools, a list of test deliverables, applicable industry standards, testing measurements and metrics, risks and mitigation, defect reporting and tracking and change/configuration management. The more specific test plan, which may live within the test strategy or as its own document, is derived from the documented business requirements and may contain test cases, conditions, scripts, testing schedules, test environments, pass/fail criteria and risk assessments.⁶ The test strategy may be developed by a project manager, with the more detailed test plan created by a test lead or team, and once completed are shared with the project team, various end users and other stakeholders for review and approval before testing begins.

Test Tools

Testing tools are widely available in the commercial market; the specific tools required will depend on the testing method(s) employed. Generally, system testing is performed either manually or through the use of automated tools. Manual testing is simply direct human interaction with a system, testing to identify defects or unexpected outcomes. A member of the test team plays the role of an end user and tests most features of the application to ensure correct behavior. To ensure completeness of testing, the test team often follows a written test plan or script that leads them through a set of important test cases. Tools required for manual testing include a written test plan, test script or scenarios to follow and a method of recording and reporting the results. Although manual testing may find many defects in a system, it is a laborious and time-consuming process. In addition, it may not be effective in finding certain classes of defects not immediately apparent to the end user.

Automated testing may be performed through the use of special software (separate from the software being tested) that controls the execution of tests, compares actual outcomes to predicted outcomes, sets up test pre-conditions and performs other test control and test reporting functions. The use of automated testing tools in healthcare is expanding, and there are many automated tools available that can be tailored specifically to an individual system's testing needs. One of the most significant benefits of test automation is the ability to duplicate the testing process. Once tests have been automated, they can quickly be run and repeated. This is often the most cost-effective method for systems that have a long maintenance life; even minor patches over the lifetime of a system can cause features to break that were working at an earlier point in time, so repeated testing is required for each patch or upgrade.²

Test Execution

Performing and documenting the test activities is the primary focus of the testing methodology. Test professionals can use any number of methods to execute test events, and most fall into one of two categories: white-box testing or black-box testing.² These approaches are based on the point of view a test engineer takes when executing test cases. White-box testing (also known as clear-box testing, glass-box testing, transparent-box testing, or structural testing) is a method of testing the *internal structures* or workings of a system, as opposed to its functionality; the tester is not concerned with how the system is supposed to behave or function, but rather with how the system is supposed to operate on an internal level. Black-box testing (also known as functional testing) is a method of software testing that tests the *functionality* of an application, as opposed to its internal structures or workings; the tester is only aware of what the system or application is supposed to do and has no knowledge of the internal operations

of the system. A hybrid of the two approaches is known as gray-box testing, which is a combination of white-box and black-box testing approaches; the tester has some knowledge of internal structures and also understands the expected system functionalities. Gray-box testing is most useful when performing tests on existing systems that have been upgraded, patched or modified.

Test methods are classified and executed based on the level of the test or the specific objective of the test. During system development, tests are performed at specific levels of development: unit-level testing, integration testing and system testing. Test methods that are not associated with a specific level of development are classified by the testing objective, such as stress, user acceptance and regression testing.²

- *Unit testing* is performed by checking individual units of source code and sets of one or more computer program modules together with associated control data, usage procedures and operating procedures to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. Unit tests are created by programmers and white-box testers during the development process. They cannot validate overall functionality on their own but are used to ensure that individual pieces function independently.
- *Integration testing* involves combining individual software modules, applications or units and testing them as a group to identify any issues in how the integrated components interface and interact with each other. Integration testing takes as its input, modules that have been unit tested, groups them into larger aggregates, applies tests defined in an integration test plan to those aggregates and delivers as its output the integrated system ready for system testing. Integration testing can be done using any of the box methods (white, black or gray) but is best suited for gray-box testing when the tester has some knowledge of the internal code of the individual units, as well as the expected system functionality.
- *System testing* is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing is one of the most common black-box testing methods and, as such, does not require knowledge of the inner design of the code or logic. System testing combines all of the integrated components that have successfully passed integration testing with software that has been integrated with hardware and tests them as a single system. The purpose of integration testing is to detect any inconsistencies between the software units that have been integrated (called assemblages) or between any of the assemblages and the hardware, as well as the exchange of data to external applications and systems.
- *Stress testing* is a form of testing that is used to determine the stability of a given system. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. The stress test puts a greater emphasis on robustness, availability and error handling under a heavy load,

rather than on what would be considered correct operation under normal circumstances. The goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space), unusually high concurrency, or denial-of-service attacks.

- *Acceptance testing* is conducted to determine if the requirements of a specification or contract are met and to validate successful system implementation. Acceptance testing is usually created by business customers (the clients or users, so also commonly referred to as user acceptance testing or UAT) and executed prior to accepting transfer of system ownership from the developer or vendor. Acceptance testing provides confidence that the delivered system meets the business requirements of sponsors, users and other stakeholders. The acceptance test may also act as the final quality gateway through which any quality defects not previously detected may be uncovered. Provided certain additional acceptance criteria are met (e.g., security testing, supportability and maintenance standards, usability standards and standards compliance), system sponsors will normally sign off on a system as satisfying contractual requirements and deliver final payment to the vendor upon successful completion of acceptance testing. Acceptance testing is also done internally when major upgrades, patches and the like are involved. The terms *acceptance testing*, *system testing* and *integration testing* may be synonymous in some organizations and in some testing situations.
- *Regression testing* is any type of system testing that seeks to uncover new bugs or errors in an existing functional system that has been changed by implementation of patches, enhancements, or configuration changes. It is common for new issues to be uncovered through the introduction of new systems. The intent of regression testing is to ensure that a planned change in software or hardware did not introduce new faults or defects into the production environment. A common method of regression testing includes repeating previously successful tests and checking to see if program behavior has changed or previously fixed bugs have reemerged after a system change.

Test Controls

System controls are implemented to protect the confidentiality, integrity and availability of data and the overall management of a system across environments during design, development, testing and deployment. Some of the most common types of test controls include version controls (also called revision controls), security audits and change controls.

- *Version control* (or revision control) tracks and provides control over changes to source code. Software developers and testers sometimes use version control software to maintain documentation and configuration files, as well as source code. As teams design, develop and test software, it is common

for multiple versions of the same software to be running in different sites and for the software's developers to be working simultaneously on updates. Often, bugs or features of the software will be present only in certain versions due to the fixing of some problems and the introduction of new ones as the program develops. Therefore, for the purposes of locating and fixing bugs, it is vitally important to be able to retrieve and run different versions of the software to determine in which version(s) a problem occurs.

- *Security audits* are manual or automatic systematic, measurable technical assessments of a system or application. Manual assessments include interviewing staff, performing security vulnerability scans, reviewing application and operating system access controls and analyzing physical access to the systems. Automated assessments include system-generated audit reports and software that monitors and reports changes to files and settings on a system. Systems that require security audits can include personal computers, servers, network routers and switches.
- *Change control* is a formal process used to ensure that changes to a product or system are introduced in a controlled and coordinated manner. It reduces the possibility that unnecessary changes will be made to a system without forethought, introducing faults, or undoing changes made by other users. Typical activities that would call for change control are patches to software products, system configuration changes, installation of new operating systems, upgrades to network routing systems and changes to the electrical power systems supporting the infrastructure. Change control is also a means by which the number of changes in an environment at any one time is controlled.

Test Results Reporting

Test results reporting occurs throughout the testing process—not just at the conclusion of a test event. Stakeholders and sponsors may expect monthly, weekly or even daily updates on current testing status, activities, schedules and more. Test reporting can be challenging and should be planned out early in the testing process. Common challenges include tailoring test reports to your audience(s), clarifying confusion about the intent of testing, explaining how testing is actually done and understanding which testing metrics are meaningful and why. At a minimum, test reports should address the mission of the test, system(s) or application(s) covered, organizational risk of deploying the system, testing techniques, test environment, updated testing status and obstacles to testing.⁷

Final Evaluation

For most testing projects, the most important deliverable is the final evaluation report, which contains the findings, conclusions and recommendations of the system test. For successful system tests, the final evaluation should confirm

to stakeholders that the system has achieved expected results and should specifically address how those test results may affect the anticipated outcomes or benefits. For example, if the results of a system test show that implementation of the system will likely significantly increase the organization's third-party insurance collections, the cost of conducting the test would be considered a sound investment and the benefits of the test would be clear. In addition, the final evaluation report should address the most common stakeholder questions at the conclusion of a test event, including (but not limited to)

- Does the system meet our quality and performance expectations?
- Is the system ready for users?
- What can we expect when x people simultaneously use the system?
- What is our potential risk if we go live with the system now?

Final evaluations may reveal the need for specific end-user training prior to the go-live event. Lessons learned from each test event should be leveraged by the team to improve the planning, execution and evaluation of future tests. Beyond the go-live date, evaluation continues to play a critical role in a system's life cycle. Post-implementation evaluations are critical for measuring initial and long-term user satisfaction, system usability, business and patient care impacts and benefits and the system's potential for expansion or integration with other organizational systems.

Summary

A successful way to mitigate risk in deploying new or modified systems is through the development of a thorough testing and evaluation strategy. And including the right people in this strategy, at the right time, is also a critical element to the success of this phase and the overall project. Healthcare systems support workflows for a variety of clinical users and clinical specialties and may be deployed across a complex healthcare network, so testing can be complex and time consuming but is, nevertheless, essential to ensuring consistent performance that supports the delivery of safe and efficient patient care.

References

1. DAU (Defense Acquisition University). *Defense Acquisition Guidebook*. Washington, DC: DAU, 2012.
2. Wikipedia. Software testing. August 2019. http://en.wikipedia.org/wiki/Software_testing (accessed September 1, 2019).
3. NIST (National Institute of Standards and Technology). Software Errors Cost U.S. Economy \$59.5 Billion Annually. Gaithersburg, MD: NIST, 2002. http://www.abeacha.com/NIST_press_release_bugs_cost.htm (accessed September 1, 2019).